The Zeus Agent Building Toolkit

ZEUS Methodology Documentation Part V

FIPA & Zeus

Simon Thompson, simon.2.thompson@bt.com

Intelligent Systems Lab, BT

Release 1.0, April 2001

Index

1.0 INTRODUCTION

FIPA is the leading "Agent Standards" body (see http://www.fipa.org), and while many aspects of Zeus have been FIPA compliant in the past (the ACL for example) it has not been possible to deploy a FIPA platform using the Zeus toolkit without doing a lot of implementation work.
In the 1.1 and 1.2 releases of Zeus features were added that allowed a FIPA compliant platform as described in FIPA specs

- XC00070 (acl representation),
- XC00075 (IIOP transport),
- XC00084 (HTTP transport),
- XC00023 (agent management),
- XC00061 (message structure),
- XC00037 (communicative acts)
- XC000067 (message transport)
- XC000008 (SL content language)[1]

to be built and deployed by using components supplied with Zeus.
We also provide a FIPA-97 compliant IIOP transport.
It should be stressed, however, that our implementation of the specifications is based on our interpretation of them, and our best effort. We aim to take part in a number of interoperability trials (Agent Cities, FIPA bakeoffs) to develop and test our compliance, but as with all software there are bound to be many issues which we either cannot, or decide not to resolve. It is our belief that all the agent platforms will evolve toward a state of interoperability that may be somewhat removed from the specifications provided by FIPA, so if Zeus behaves in a way that differs from the specs - don't be surprised.
If however it starts emitting error messages then there is probably a bug that should be reported in the normal way (zeus@jiscmail.ac.uk)

2.0 COMPONENTS OF A FIPA PLATFORM

We have implemented:

- An Agent Communication Channel (ACC). This provides message transports for FIPA2000 IIOP, FIPA'97 IIOP and FIPA 2000 HTTP. It also manages communication between agents on a Zeus platform and agents on another FIPA platform. We give a more detailed description of the ACC in section 2.3 of this document.
- A Directory Facilitator (DF). This provides a searchable directory of agent services. We give a description of the DF in section 2.4 of this document.
- An Agent Management Service (AMS). The AMS is described in section 2.5 of this document.

Our model is shown in figure 1 below.

[1] using the JADE LGPL SL parser. Note, limited support only.

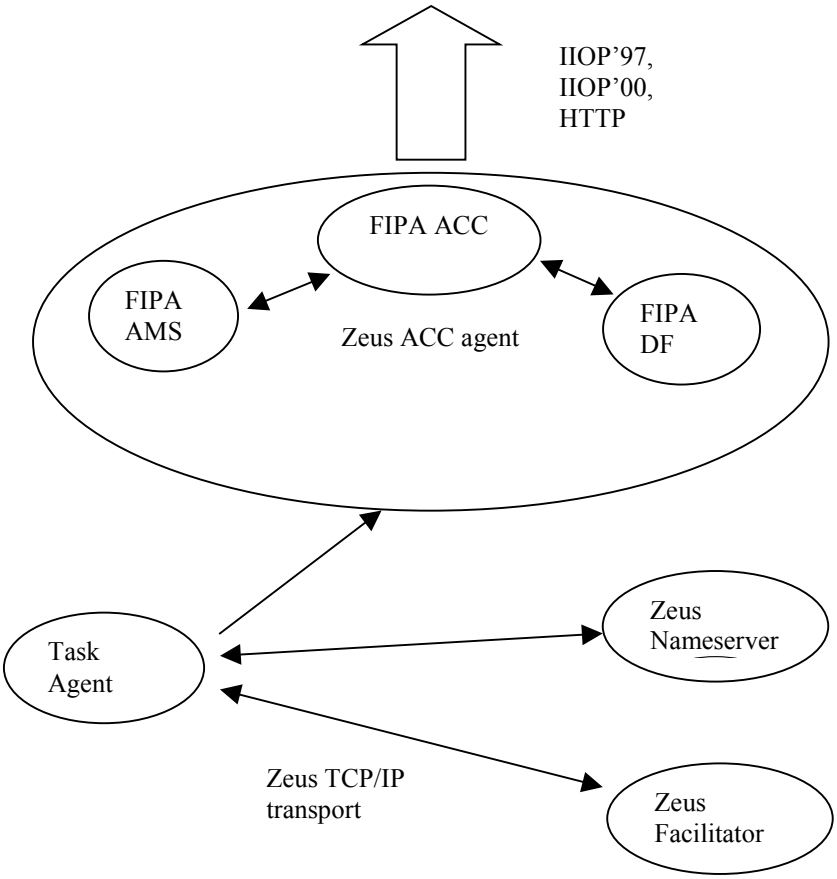

Figure 1. Basic FIPA Zeus architecture

2.1 Running the FIPA agent platform

All of the FIPA platform components described above are implemented in the 1.2 release in the form of one Zeus agent (as shown in figure 1.), the Zeus ACC agent.
To run a FIPA platform you must ensure the following commands have been issued on the chosen host machine.

```
start /min java zeus.agents.ANServer Nameserver4 -t 0.5 -f
dns.db
```

which runs the Zeus nameserver.

```
start /min tnameserv -ORBInitialPort 2000
start /min tnameserv -ORBInitialPort 1097
```

Which are used to run the java orbs (object request brokers) for IIOP communication. The command `tnameserv` is the command that invokes the java orb, the parameter `ORBInitialPort` is used to set the port that the orb will be listening on. For the default implementation that we have supplied these are set to 2000 and 1097 for the IIOP 2000 and IIOP 1997 transports respectively (for obvious reasons). These numbers can be set to different values as explained in section 2.2.

```
start /min java zeus.agents.ACC -s dns.db -o null -fipaNames
contacts.txt -transports config.txt -gui
zeus.agentviewer.AgentViewer
```

The command `java zeus.agents.ACC` invokes the Java virtual machine to run the main method of the class zeus.agents.ACC, which is the agent we want to run. The parameter `-s dns.db` tells the ACC agent to use the nameserver database that it finds in its directory, `-o null` tells the ACC agent not to bother loading an ontology, `-fipaNames contacts.txt` and `-transports config.txt` tells the agent to load platform configuration information from the two files contracts.txt and config.txt as explained in section 2.2. Finally, the parameter `-gui zeus.agentviewer.AgentViewer` is used to ask the ACC agent to start an agentviewer tool so that it's internal workings can be viewed by the platform owner.
I am unsure whether or not having an agentviewer tool started with the ACC agent is a good or bad thing. On the onehand it represents a definite overhead to the system at a fairly cruicial bottleneck, on the other hand it is clear that being able to see what the ACC is up to could be very handy. Perhaps readers could feed there views back onto the Zeus mail list or to me personally.

Other agents can be started on the platform in the normal way (see the Zeus Runtime Guide for help).

2.2 Configuring the Platform

As described above there are two files that are used uniquely with the ACC agent and can be used to configure it.

config.txt

config.txt is used to provide configuration information to the Zeus ACC so that it can start its inboxes (normally via the class zeus.actors.intrays.Zeus_ACC_Server) on the port numbers desired by the platform owner.

FIPA addresses have the form

```
xxxx://111.222.333.444:port/name1/name2/.../namen
```

where xxxx is the name of the transport ie. iiop or http; 111.222.333.444 is a ip address , port is the port number on the receiveing machine that is to be used and the names are the names of the objects that are to be accessed.
In the FIPA Zeus implementation the names of the agents providing platform functionality are predefined. Only the name acc is used, and all communication from external platforms is routed through this address. We have not provided any mechanism to allow this to be customised easily at this time, although later versions may do so.
However, the port numbers of the addresses to be used are configurable. Config.txt should contain a pseudo-xml string of the form

```
<transport_configuration>
    <name> FIPA_IIOP_2000 </name>
    <port> 2000 </port>
</transport_configuration>
<transport_configuration>
    <name> FIPA_IIOP_1997 </name>
    <port> 1997 </port>
</transport_configuration>
<transport_configuration>
    <name> FIPA_HTTP_2000 </name>
    <port> 16100 </port>
</transport_configuration>
```

By altering the numbers in the <port> tags you can alter the port that the ACC agent will listen on for that particular transport. For example, if you need a FIPA 2000 IIOP service running on

port number 2001, an FIPA 1997 service on port number 1998 and a HTTP service on port number 80 create a config.txt file as below:

```
<transport_configuration>
    <name> FIPA_IIOP_2000 </name>
    <port> 2001 </port>
</transport_configuration>
<transport_configuration>
    <name> FIPA_IIOP_1997 </name>
    <port> 1998 </port>
</transport_configuration>
<transport_configuration>
    <name> FIPA_HTTP_2000 </name>
    <port> 80 </port>
</transport_configuration>
```

It is important to retain the spaces around the tags in the file as the parser that I implemented for this is not very good, and will crash if it doesn't see whitespace after each > character.

contacts.txt

contacts.txt is used to bootstrap the connection between Zeus and another platform.

When the ACC receives a message from another agent that it is required to forward onto the Zeus platform it will record the address of the sending agent and register itself as a proxy in the nameserver. This functionality is described in detail in section 2.3. The important point for the purposes of this section is to note that unless a message is received, or the address of the agent on another platform is explictly programmed (for example by setting up a register message as in the ams_df test case) the platform has no way of knowing where to send messages to an agent on another platform!

contacts.txt is used to overcome this. Agent addresess are coded into it using the following format:

```
<alias_address>
<zeus_name>
    ping_agent
</zeus_name>
<aid>
 (agent-identifier
      :name PingAgent@132.146.209.235
      :addresses (sequence
iiop://132.146.209.235:1097/acc))
</aid>
</alias_address>
```

The element <zeus_name> contains the alias that the remote agent is to be registered under on the local nameserver. The element <aid> contains the address that the ACC is to use to contact PingAgent. If an alias like this is set into the contacts.txt file then any agent on the Zeus platform can communicate with the remote agent simply by sending it a normal Zeus performative with the sender field set to the alias value (in this case "ping_agent").

2.3 The ACC

The ACC agent is the Agent Communication Channel as specified by FIPA. It can be thought of as the single point of contact for all agents on a platform.
The basic function of the ACC is to forward messages to the other agents, and to accept messages for forwarding from agents on it's platform and pass them on.

The ACC in this release of Zeus (1.2) provides intrays and outtrays for three transports -

- the FIPA'97 IIOP transport, which is provided by a CORBA interface implementing the fipa_97_agent class;
- the FIPA 2000 IIOP agent which is provided by a CORBA interface implementing FIPA.mts;
- the FIPA 2000 HTTP class which is implemented by sockets that listen over TCP/IP for connections from remote machines

The ACC in Zeus also has the function of providing translation facilities between Zeus performatives and FIPA performatives, in particular resolving FIPA addresses to Zeus addresses (and vica versa) and providing FIPA envelopes.

Finally the ACC is responsible for hiding the complexities of FIPA addressing from client Zeus agents. When a message from outside Zeus is received by the ACC it follows the following procedure

- parse message into FIPA storage objects
- generate unique alias for agent (ensure that there is only one alias per external agent)
- register self under alias with nameserver
- set up message handling rule for messages for alias in self.
- translate message to Zeus performative and set sender as alias
- send message to appropriate agent.

When a Zeus agent replies to the message it will use the sender as the reply address (the reply-to field should also be set to the alias). Because it will have no actual address for the alias in its local address book the agent will ask the nameserver for the address of the alias agent. The nameserver will supply the ACC's address. When the message is sent to the ACC it will invoke the message handling rule for the alias and translate the message into a FIPA performative before sending it to the address that it received the original message from.

2.4 DF

The DF is the FIPA service lookup object.

The DF supplied with Zeus 1.2 provides basic register, deregister and search functionality. Services must be explicitly registered with the DF to be accessed from external platforms.

In future releases of Zeus we plan to provide an option that will allow the registration of services provided by Zeus primitive and rule based tasks in the FIPA DF, because the String/SL representation for services allows a more sophisticated service model than the current Zeus model.

2.5 AMS

The AMS is the Agent Management Service.

The AMS provided by Zeus 1.1 implements register, deregister, search and get-description functionality. FIPA AMS services are fairly crude and we have no plans to make any further use of them, apart from possibly supporting automatic registration of deployed agents in the FIPA address space.

3.0 SL

SL is the content language mandated by FIPA.

The SL/Agent Management Ontology support provided by Zeus is implemented by a bespoke parser (FIPAParser.jj), and is not suited for general used.

However we have “adopted” the SL parser used by JADE, and bundled it with the 1.2 distribution of Zeus. This software is released under the LGPL license, and so it is possible to redistribute it in this way. However, I would like to stress that all credit for the classes in the package sl; should be directed to the JADE team, especially Fabio Bellifemine and Giovanni Rimassa who wrote it.

The reason that I have done this is quite simple: I don’t expect that SL will be the content language of choice for agent applications of the future, and so I can’t be bothered to write my own parser for it. However, this view is largely based on ignorance, and it is possible that we will extend SL support in future releases of Zeus.